UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/929,448	08/14/2001	Christian Linhart	218.1022	8783

23280 7590 07/03/2008
Davidson, Davidson & Kappel, LLC
485 7th Avenue
14th Floor
New York, NY 10018

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
07/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/929,448	LINHART, CHRISTIAN	
	Examiner	Art Unit	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 25 January 2008.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-88 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-88 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 28 January 2008 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated January 25, 2008, responding to the Office action mailed July 26, 2007 provided in the rejection of claims 1-88, wherein claims 6, 64 and 70 were amended.

Claims 1-88 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see *Liao* - art made of record, as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1, 3-47, 49-64, 66-72, and 74-88 are rejected under 35 U.S.C. 103(a) as being unpatentable over Murphy et al., (*Lightweigh Lexical Source Model Extraction, July 1996, ACM*) (hereinafter 'Murphy') in view of Vern Paxon, (*Flex- Fast Lexical Analyzer Generator, June 1998, The Free Software Foundation*) (hereinafter 'Paxon')

and further in view of Heng Liao (Pat. No. US 7,188,168 B1) (hereinafter 'Liao' - art made of record)

3. **As to claim 1** (Original), Murphy discloses a method for generating a scanner (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 2nd Par. – our system consists of two generators: a scanner generator and an analyzer generator), the method comprising: receiving a definition of a plurality of patterns; receiving a definition of a respective association between each of the plurality of patterns and a respective executable action (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output)

Murphy discloses the pattern and action definitions (e.g., P. 266, 1st Par. – the scanner generator uses the first two parts – the pattern and action definitions to generate a scanner that reads in a sequence of artifact files and produces a stream of local output), but does not explicitly disclose processing the plurality of patterns and the respective associations to form a scanner data structure capable of comparing input data to at least one of the plurality of patterns and causing execution of the associated executable action upon a match of the input data with the respective one of the plurality of patterns.

However, in an analogous art of *flex – fast lexical analyzer generator*, Paxon discloses processing the plurality of patterns and the respective associations to form a scanner data structure capable of comparing input data to at least one of the plurality of

patterns and causing execution of the associated executable action upon a match of the input data with the respective one of the plurality of patterns (e.g., P. 7, 3rd Par. – the rules section of the flex input contains <u>a series of rules of the form: *pattern action*</u>; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that <u>*yytext*</u> can defined in two different ways: either <u>as a character pointer</u> or <u>as a character array – via index</u>)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Paxon into the Murphy's system to further provide processing the plurality of patterns and the respective associations to form a scanner data structure capable of comparing input data to at least one of the plurality of patterns and causing execution of the associated executable action upon a match of the input data with the respective one of the plurality of patterns in Murphy system.

The motivation is that it would further enhance the Murphy's system by taking, advancing and/or incorporating Paxon's system which offers significant advantages for

fast lexical analyzer generator; interfacing with *Yacc*; performance; generating C++ scanners, and incompatibilities with **Lex** and *Posix* etc., as once suggested by Paxon (e.g., P. 1 through P. 3)

Further, Murphy and Paxon do not explicitly disclose the processing and the comparing being performed in a same active process.

However, in an analogous art of *Method and Apparatus for Grammatical Packet Classifier*, Liao discloses the processing and the comparing being performed in a same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liao into the Murphy-Paxon's system to further provide the processing and the comparing being performed in a same active process in Murphy-Paxon system.

The motivation is that it would further enhance the Murphy-Paxon's system by taking, advancing and/or incorporating Liao's system which offers significant advantages that the invention in one aspect adapts lexical token scanning techniques developed for compiler to data communications and can recognize the internal structure of packets (e.g., network data packets - viewed as a language) in real time as once suggested by Liao (e.g., Col. 3, Lines 15-21, 54-62; Col. 5, Lines 62-67)

4. **As to claim 3** (Original) (incorporating the rejection in claim 1), Liao discloses the receiving the definition of the plurality of patterns is performed in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string).

5. **As to claim 4** (Original) (incorporating the rejection in claim 1), Murphy discloses plurality of patterns include regular expressions (e.g., Sec. 1 – Introduction, 2nd Par. –

an engineer produces a source model with these tools by specifying regular expressions to be matched to text within the artifacts).

6. **As to claim 5** (Original) (incorporating the rejection in claim 1), Murphy discloses plurality of patterns include text patterns (Sec. 2 – The Lexical Source Model Extraction Approach, 2nd Par., Item 1 – patterns describing constructions of interest in a system artifact; Sec. 3.1 – Specifying Patterns, 1st Par. – The engineer describes the information to extract from the system artifacts as a set of hierarchical patterns; each patter uses regular expression to describe a construct that may be found within the artifacts).

7. **As to claim 6** (Currently Amended) (incorporating the rejection in claim 1), Liao discloses receiving the definition of the respective association is performed in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that <u>integrates the lexical scanner and the parsing engine</u>. When the labeled DFA scans the input data, t<u>he lexical state and the parsing states are updated at the same time</u>. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a <u>subset construction algorithm</u> 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table …

The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

8. **As to claim 7** (Original) (incorporating the rejection in claim 1), Paxon discloses each respective association between each of the plurality of patterns and the respective executable action includes a pointer (e.g., P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

9. **As to claim 8** (Original) (incorporating the rejection in claim 1), Paxon discloses each respective association between each of the plurality of patterns and the respective executable action includes an index number (e.g., P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

10. **As to claim 9** (Original) (incorporating the rejection in claim 1), Murphy discloses at least one of the respective executable action includes a respective action object (e.g., Sec. 3 – The Specification Language – action code to execute after a pattern is matched to a portion of a system artifact)

11. **As to claim 10** (Original) (incorporating the rejection in claim 1), Liao discloses at least a portion of at least one of the respective executable action is generated in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62

through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

12. **As to claim 11** (Original) (incorporating the rejection in claim 1), Murphy discloses at least one of the respective executable action includes respective program code (e.g., Sec. 3 – The Specification Language – action code to execute after a pattern is matched to a portion of a system artifact)

13. **As to claim 12** (Original) (incorporating the rejection in claim 1), Murphy discloses wherein at least one of the respective executable action includes respective data (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 5th Par., Lines 1-5 – two scanners: on to scan C source code and another to scan structured data files)

14. **As to claim 13** (Original) (incorporating the rejection in claim 1), Paxon discloses the processing includes forming a scanner definition string from the plurality of patterns

and respective associations (e.g., P. 7, 3rd Par. – the rules section of the flex input contains <u>a series of rules of the form: *pattern action*</u>; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action)

15. **As to claim 14** (Original) (incorporating the rejection in claim 13), Paxon discloses that each respective association is represented in the scanner definition string by a respective indicator (e.g., P. 13, 2nd Par. – Note that <u>*yytext*</u> can defined in two different ways: either <u>as a character pointer</u> or <u>as a character array – via index</u>)

16. **As to claim 15** (Original) (incorporating the rejection in claim 14), Paxon discloses that the processing further includes saving a mapping between each respective indicator and a respective action-pointer (Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that <u>*yytext*</u> can defined in two different ways: either <u>as a character pointer</u> or <u>as a character array – via index</u>)

17. **As to claim 16** (Original) (incorporating the rejection in claim 14), Paxon discloses that each respective indicator includes a respective index (e.g., P. 13, 2nd Par.

– Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

18. **As to claim 17** (Original) (incorporating the rejection in claim 14), Paxon discloses that each respective indicator includes a respective a number representing a respective pointer (e.g., P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

19. **As to claim 18** (Original) (incorporating the rejection in claim 13), Paxon discloses that the scanner definition string has a format of a scanner definition file of a scanner generator (e.g., P. 8, Patterns – the patterns in the input are written using an extended set of regular expressions)

20. **As to claim 19** (Original) (incorporating the rejection in claim 18), Paxon discloses wherein the scanner generator is a *Flex* scanner generator (e.g., P. 1, Name, flex – fast lexical analyzer generator; P. 3, Description)

21. **As to claim 20** (Original) (incorporating the rejection in claim 18), Paxon discloses wherein the processing further includes inputting the scanner definition string into a scanner generator core modified for processing the scanner definition string so as to form a processed scanner definition data structure (e.g., P. 7, 3rd Par. – the rules section of the flex input contains a series of rules of the form: *pattern action*;

PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

22. **As to claim 21** (Original) (incorporating the rejection in claim 20), Paxon discloses wherein the scanner generator core is a *Flex* scanner generator core (e.g., P. 1, Name, flex – fast lexical analyzer generator; P. 3, Description)

23. **As to claim 22** (Original) (incorporating the rejection in claim 20), Paxon discloses wherein the processing further includes converting the processed scanner definition data structure into the scanner data structure (e.g., P. 7, 3rd Par. – the rules section of the flex input contains a series of rules of the form: *pattern action*; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the

generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

24. **As to claim 23** (Original) (incorporating the rejection in claim 1), Paxon discloses wherein the scanner data structure includes a respective indicator representing the respective association between each of the plurality of patterns and the respective executable action (e.g., P. 7, 3rd Par. – the rules section of the flex input contains a series of rules of the form: *pattern action*; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. –

Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

25. **As to claim 24** (Original) (incorporating the rejection in claim 23), Paxon discloses wherein each respective indicator includes an index (e.g., P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

26. **As to claim 25** (Original) (incorporating the rejection in claim 23), Paxon discloses wherein each respective indicator includes a pointer (e.g., P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

27. **As to claim 26** (Original) (incorporating the rejection in claim 13), Paxon discloses wherein the processing further includes saving the scanner definition string and the scanner data structure (e.g., P. 3, Description – *flex* is a tool for generating scanners: programs which recognized lexical patterns in text; *flex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate)

28. **As to claim 27** (Original) (incorporating the rejection in claim 26), Paxon discloses wherein the saving is performed to a persistent memory device (e.g., P. 3, Description – *flex* is a tool for generating scanners: programs which recognized lexical

patterns in text; *flex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate)

29. **As to claim 28** (Original) (incorporating the rejection in claim 26), Paxon discloses wherein the processing further includes saving a mapping between the scanner definition string and the scanner data structure (e.g., P. 3, Description – *flex* is a tool for generating scanners: programs which recognized lexical patterns in text; *flex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate)

30. **As to claim 29** (Original) (incorporating the rejection in claim 28), please refer to claim **27** as set forth above accordingly.

31. **As to claim 30** (Original) (incorporating the rejection in claim 28), Murphy discloses further comprising: receiving a definition of a plurality of second patterns; receiving a definition of a respective second association between each of the plurality of second patterns and a respective second executable action; processing the plurality of second patterns and the respective second associations so as to determine a second scanner data structure capable of comparing the input data to each of the plurality of second patterns and causing execution of the respective second executable action upon a match of the input data with a one of the plurality of second patterns, the processing each of the plurality of second patterns and the respective second

association and the comparing the input data to each of the plurality of second patterns being performed in a same second active process; wherein the processing each of the plurality of second patterns and the respective second association includes: forming a second scanner definition string from the plurality of second patterns and respective second associations; and comparing the second scanner definition string to the saved scanner definition string and loading the saved scanner data structure as the second scanner data structure when the second scanner definition string matches the saved scanner definition string (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

32. **As to claim 31** (Original) (incorporating the rejection in claim 30), Murphy discloses wherein the saved scanner definition string and the second scanner definition string have a format of a scanner definition file of a scanner generator (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures

scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

33. **As to claim 32** (Original) (incorporating the rejection in claim 31), please refer to claim **19** as set forth above accordingly.

34. **As to claim 33** (Original) (incorporating the rejection in claim 30), Murphy discloses wherein the loading the saved scanner data structure as the second scanner data structure is performed using the mapping between the scanner definition string and the scanner data structure (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

35. **As to claim 34** (Original) (incorporating the rejection in claim 30), Murphy discloses further comprising inputting the second scanner definition string into a

scanner generator core so as to form a processed scanner definition data structure when the second scanner definition string does not match the saved scanner definition string (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6))

36. **As to claim 35** (Original) (incorporating the rejection in claim 34), please refer to claim **19** as set forth above accordingly.

37. **As to claim 36** (Original) (incorporating the rejection in claim 34), Murphy discloses wherein the processing further includes converting the processed scanner definition data structure into the second scanner data structure (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second

pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

38. **As to claim 37** (Original) (incorporating the rejection in claim 30), Murphy discloses wherein the processing further includes saving the second scanner definition string and the second scanner data structure (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information – the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

39. **As to claim 38** (Original) (incorporating the rejection in claim 37), Murphy discloses wherein the processing further includes saving a mapping between the second scanner definition string and the second scanner data structure (e.g., P. 268, 1st Par – during extraction, the source will be scanned for an occurrence of the first patter; once the first pattern is matched, scanning will continue looking for both another occurrence of the first pattern and occurrence of the second pattern; this ensures scanning will not miss the start of the next function declaration while still being tolerant to syntactical deviations in the source code; P. 276, Sec. of Scanning Event Information

– the second pattern matches calls within a function body that take a constant character string as a first argument; action code id depicted within boxes in Fig. 6)

40. **As to claim 39** (Original) (incorporating the rejection in claim 1), Murphy discloses further comprising: associating at least one respective start state of a plurality of start states with each of the plurality of patterns; and setting a current start state, the current start state being one of the plurality of start states; wherein the processing includes processing the at least one respective start state along with each associated pattern so as to form a part of the scanner data structure, the comparing being performed so as to compare only the patterns of the plurality of patterns associated with a respective start state equal to the current start state (e.g., P. 273, Sec. of Translating Patterns to State Machines – we ensure the correct linkage by first performing a straightforward translation from a given pattern (i.e., regular expression) into a nondeterministic finite-state machine with e moves)

41. **As to claim 40** (Original) (incorporating the rejection in claim 39), Paxon discloses wherein the setting the current start state is performed so as to reset the current start state to another one of the plurality of start states at least once during the comparing (e.g., P. 20, Sec. of Start Conditions. – flex provides a mechanism for conditionally activating rules; any rule whose pattern is prefixed with "<sc>" will only be active when the scanner is in the start condition named "sc")

42. **As to claim 41** (Original) (incorporating the rejection in claim 40), Paxon discloses wherein the current start state is reset using at least one of the respective executable action (P. 20, Sec. of Start Conditions. – flex provides a mechanism for conditionally activating rules; any rule whose pattern is prefixed with "<sc>" will only be active when the scanner is in the start condition named "sc")

43. **As to claim 42** (Original) (incorporating the rejection in claim 40), Paxon discloses further comprising maintaining a stack of the plurality of start states, the current start state being the top of the stack (e.g., P. 27, 3rd Par. – pops the top of the stack and switches to it via BEGIN; returns the top of the stack without altering the stack's contents)

44. **As to claim 43** (Original) (incorporating the rejection in claim 39), Paxon discloses further comprising associating a respective context with each of the plurality of start states for controlling the respective start state (e.g., P. 2 – Start Conditions – introducing context into your scanners, and managing "mini-scanners"; P. 12, 2nd Par. – a rule can have at most one instance of trailing context; P. 12, Sec. of How The Input Is Matched; if it finds more than one match, it takes the one matching the most text (for trailing context rules, this includes the length of the trailing part, even thought it will then be returned to the input); if it finds two or more matches of the same length, the rule listed first in the *flex* input file is chosen)

45. **As to claim 44** (Original) (incorporating the rejection in claim 43), Paxon discloses wherein each context includes at least one respective rule defining a start and an end of the context (e.g., P. 2 – Start Conditions – introducing context into your scanners, and managing "mini-scanners"; P. 12, 2nd Par. – a rule can have at most one instance of trailing context; P. 12, Sec. of How The Input Is Matched; if it finds more than one match, it takes the one matching the most text (for trailing context rules, this includes the length of the trailing part, even thought it will then be returned to the input); if it finds two or more matches of the same length, the rule listed first in the *flex* input file is chosen)

46. **As to claim 45** (Original) (incorporating the rejection in claim 39), Paxon discloses comprising maintaining a stack of the plurality of start states, the current start state being the top of the stack, each of the context for controlling a position of the respective start state in the stack (e.g., P. 27, 3rd Par. – three routines are available for manipulating stacks of start conditions: *yy_push_state(), yy_pop_state(), and yy_top_state()*)

47. **As to claim 46** (Original) (incorporating the rejection in claim 1), Murphy discloses wherein the input data includes a data stream (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 5th Par., Lines 1-5 – two scanners: on to scan C source code and another to scan structured data files)

48. **As to claim 47** (Original), Murphy discloses a method for scanning input data, the method comprising: receiving a definition of a plurality of patterns; receiving a definition of a respective association between each of the plurality of patterns and a respective executable action e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output).

Murphy discloses the pattern and action definitions (P. 266, 1st Par. – the scanner generator uses the first two parts – the pattern and action definitions to generate a scanner that reads in a sequence of artifact files and produces a stream of local output), but does not explicitly disclose processing the plurality of patterns and the respective associations so as form a scanner data structure; comparing the input data to the a plurality of patterns using the scanner data structure; and when the comparing results in a matched one of the a plurality of patterns, executing the respective executable action associated with the matched pattern.

However, in an analogous art of flex – fast lexical analyzer generator, Paxon discloses processing the plurality of patterns and the respective associations so as form a scanner data structure; comparing the input data to the a plurality of patterns using the scanner data structure; and when the comparing results in a matched one of the a plurality of patterns, executing the respective executable action associated with the matched pattern (e.g., P. 7, 3rd Par. – the rules section of the flex input contains <u>a series of rules of the form: *pattern action*</u>; PATTERNS – P. 8 through P. 12, the patterns in the

input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Paxon into the Murphy's system to further provide processing the plurality of patterns and the respective associations so as form a scanner data structure; comparing the input data to the a plurality of patterns using the scanner data structure; and when the comparing results in a matched one of the a plurality of patterns, executing the respective executable action associated with the matched pattern in Murphy system.

The motivation is that it would further enhance the Murphy's system by taking, advancing and/or incorporating Paxon's system which offers significant advantages for fast lexical analyzer generator; interfacing with *Yacc*; performance; generating C++ scanners, and incompatibilities with *Lex* and *Posix* etc., as once suggested by Paxon (e.g., P. 1 through P. 3)

Further, Murphy and Paxon do not explicitly disclose the processing and the comparing are performed in a same active process.

However, in an analogous art of *Method and Apparatus for Grammatical Packet Classifier*, Liao discloses the processing and the comparing are performed in a same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liao into the Murphy-Paxon's system to further provide the processing and the comparing are performed in a same active process in Murphy-Paxon system.

The motivation is that it would further enhance the Murphy-Paxon's system by taking, advancing and/or incorporating Liao's system which offers significant advantages that the invention in one aspect adapts lexical token scanning techniques developed for compiler to data communications and can recognize the internal structure

of packets (e.g., network data packets - viewed as a language) in real time as once suggested by Liao (e.g., Col. 3, Lines 15-21, 54-62; Col. 5, Lines 62-67)

49. **As to claim 48** (Original) (incorporating the rejection in claim 47), please refer to claim **2** as set forth above accordingly.

50. **As to claim 49** (Original) (incorporating the rejection in claim 47), please refer to claim **3** as set forth above accordingly.

51. **As to claim 50** (Original) (incorporating the rejection in claim 47), Liao discloses wherein the receiving a definition of the respective association between each of the plurality of patterns and the respective executable action is performed in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

52. **As to claim 53** (Original) (incorporating the rejection in claim 47), Murphy discloses wherein the processing includes forming a scanner definition string from the plurality of patterns and respective associations (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output)

53. **As to claim 54** (Original) (incorporating the rejection in claim 47), please refer to claim **20** as set forth above accordingly.

54. **As to claim 55** (Original) (incorporating the rejection in claim 54), please refer to claim **22** as set forth above accordingly.

55. **As to claim 56** (Original) (incorporating the rejection in claim 53), please refer to claim **26** as set forth above accordingly.

56. **As to claim 57** (Original) (incorporating the rejection in claim 56), please refer to claim **28** as set forth above accordingly.

57. **As to claim 58** (Original) (incorporating the rejection in claim 57), please refer to claim **30** as set forth above accordingly.

58. **As to claim 59** (Original) (incorporating the rejection in claim 58), please refer to claim **33** as set forth above accordingly.

59. **As to claim 60** (Original) (incorporating the rejection in claim 58), please refer to claim **34** as set forth above accordingly.

60. **As to claim 61** (Original) (incorporating the rejection in claim 47), please refer to claim **39** as set forth above accordingly.

61. **As to claim 62** (Original) (incorporating the rejection in claim 61), please refer to claim **43** as set forth above accordingly.

62. **As to claim 63** (Original) (incorporating the rejection in claim 62), please refer to claim **44** as set forth above accordingly.

63. **As to claim 64** (Currently Amended), Murphy discloses a scanner comprising: a scanner data structure stored on a computer readable medium including processed information for a plurality of patterns and respective indicators for associating a respective executable action with each of the a plurality of patterns (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output).

Murphy discloses the pattern and action definitions (e.g., P. 266, 1st Par. – the scanner generator uses the first two parts – the pattern and action definitions to

generate a scanner that reads in a sequence of artifact files and produces a stream of local output), but does not explicitly disclose the scanner data structure being capable of being used to compare input data to the plurality of patterns and cause execution of the respective executable action upon a match of the input data with a one of the a plurality of patterns.

However, in an analogous art of *flex – fast lexical analyzer generator*, Paxon discloses the scanner data structure being capable of being used to compare input data to the plurality of patterns and cause execution of the respective executable action upon a match of the input data with a one of the a plurality of patterns (e.g., P. 7, 3rd Par. – the rules section of the flex input contains a series of rules of the form: *pattern action*; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Paxon into the Murphy's

system to further provide the scanner data structure being capable of being used to compare input data to the plurality of patterns and cause execution of the respective executable action upon a match of the input data with a one of the a plurality of patterns in Murphy system.

The motivation is that it would further enhance the Murphy's system by taking, advancing and/or incorporating Paxon's system which offers significant advantages for fast lexical analyzer generator; interfacing with *Yacc*; performance; generating C++ scanners, and incompatibilities with *Lex* and *Posix* etc., as once suggested by Paxon (e.g., P. 1 through P. 3)

Further, Murphy and Paxon do not explicitly disclose the scanner data structure being formed and the comparing being performed in a same active process.

However, in an analogous art of *Method and Apparatus for Grammatical Packet Classifier*, Liao discloses the scanner data structure being formed and the comparing being performed in a same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that <u>integrates the lexical scanner and the parsing engine</u>. When the labeled DFA scans the input data, t<u>he lexical state and the parsing states are updated at the same time</u>. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a <u>subset construction algorithm</u> 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … <u>The transition table is constructed</u> so

that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liao into the Murphy-Paxon's system to further provide the scanner data structure being formed and the comparing being performed in a same active process in Murphy-Paxon system.

The motivation is that it would further enhance the Murphy-Paxon's system by taking, advancing and/or incorporating Liao's system which offers significant advantages that the invention in one aspect adapts lexical token scanning techniques developed for compiler to data communications and can recognize the internal structure of packets (e.g., network data packets - viewed as a language) in real time as once suggested by Liao (e.g., Col. 3, Lines 15-21, 54-62; Col. 5, Lines 62-67)

64. **As to claim 66** (Original) (incorporating the rejection in claim 64), please refer to claim **3** as set forth above accordingly.

65. **As to claim 67** (Original) (incorporating the rejection in claim 64), Liao discloses wherein a definition of each of the respective indicator is received in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing

states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

66. **As to claim 68** (Original) (incorporating the rejection in claim 64), Murphy discloses wherein at least one of the respective executable action includes a respective action object (e.g., Sec. 3 – The Specification Language – action code to execute after a pattern is matched to a portion of a system artifact)

67. **As to claim 69** (Original) (incorporating the rejection in claim 64), Liao discloses the scanner wherein at least a portion of at least one of the respective executable action is generated in the same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a

number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

68. **As to claim 70** (Currently Amended) (incorporating the rejection in claim 64), Murphy discloses the scanner wherein the scanner definition data structure set in a computer readable medium is formed by a process including forming a scanner definition string from the plurality of patterns and respective associations (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output)

69. **As to claim 71** (Original) (incorporating the rejection in claim 70), Murphy discloses the scanner wherein the process further includes inputting the scanner definition string into a scanner generator core modified for processing the scanner definition string (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output)

70. **As to claim 72** (Original), Murphy discloses a computer readable medium having stored thereon computer executable process steps operative to perform a method for generating a scanner (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 2nd

Par. – our system consists of two generators: a scanner generator and an analyzer generator), the method comprising: defining a plurality of patterns; defining a respective association between each of the plurality of patterns and a respective executable action (e.g., Sec. 2 – The Lexical Source Model Extraction Approach, 3rd Par. – the scanner generator uses the first two parts – the pattern and action definitions – to generate a scanner that reads in a sequence of artifact files and produces a stream of local output).

Murphy discloses the pattern and action definitions (P. 266, 1st Par. – the scanner generator uses the first two parts – the pattern and action definitions to generate a scanner that reads in a sequence of artifact files and produces a stream of local output), but does not explicitly disclose processing the plurality of patterns and the respective associations so as to form a scanner data structure capable of comparing input data to each of the plurality of patterns and causing execution of the respective executable action upon a match of the input data with a one of the plurality of patterns.

However, in an analogous art of *flex – fast lexical analyzer generator*, Paxon discloses processing the plurality of patterns and the respective associations so as to form a scanner data structure capable of comparing input data to each of the plurality of patterns and causing execution of the respective executable action upon a match of the input data with a one of the plurality of patterns (e.g., P. 7, 3rd Par. – the rules section of the flex input contains <u>a series of rules of the form: *pattern action*</u>; PATTERNS – P. 8 through P. 12, the patterns in the input are written using an extended set of regular expressions; ACTIONS – P. 13 through P. 18, each pattern in a rule has a corresponding action; HOW THE INPUT IS MATCHED – P. 12 through P. 13, when the

generated scanner is run, it analyzes its input looking for strings which match any of its patterns; P. 12, Sec. of How The Input Is Matched, 2nd Par. – Once the match is determined, the text corresponding to the match (called the *token*) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*; the action corresponding to the matched pattern is then executed, and then the remaining input is canned for another match; P. 13, 2nd Par. – Note that *yytext* can defined in two different ways: either as a character pointer or as a character array – via index)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Paxon into the Murphy's system to further provide processing the plurality of patterns and the respective associations so as to form a scanner data structure capable of comparing input data to each of the plurality of patterns and causing execution of the respective executable action upon a match of the input data with a one of the plurality of patterns in Murphy system.

The motivation is that it would further enhance the Murphy's system by taking, advancing and/or incorporating Paxon's system which offers significant advantages for fast lexical analyzer generator; interfacing with *Yacc*; performance; generating C++ scanners, and incompatibilities with *Lex* and *Posix* etc., as once suggested by Paxon (e.g., P. 1 through P. 3)

Further, Murphy and Paxon do not explicitly disclose the processing and the comparing being performed in a same active process.

However, in an analogous art of *Method and Apparatus for Grammatical Packet Classifier*, Liao discloses the processing and the comparing being performed in a same active process (e.g., Fig. 1, element 116 – Subset Construction; Col. 6, Line 62 through Col. 7, Line 5 - … an enhanced sub-set construction algorithm is then applied to the combined NFA to generate a labeled DFA that integrates the lexical scanner and the parsing engine. When the labeled DFA scans the input data, the lexical state and the parsing states are updated at the same time. The DFA state represent the lexical state, and the DFA labels identify the parsing engine states; Col. 9, Lines 26-32 – a subset construction algorithm 116 is used to convert the labeled NFA L into an equivalent labeled DFA. The algorithm outputs a number of DFA states and a transition table … The transition table is constructed so that the DFA will simulate in parallel all possible moves the NFA can make on a given input string)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liao into the Murphy-Paxon's system to further provide the processing and the comparing being performed in a same active process in Murphy-Paxon system.

The motivation is that it would further enhance the Murphy-Paxon's system by taking, advancing and/or incorporating Liao's system which offers significant advantages that the invention in one aspect adapts lexical token scanning techniques developed for compiler to data communications and can recognize the internal structure of packets (e.g., network data packets - viewed as a language) in real time as once suggested by Liao (e.g., Col. 3, Lines 15-21, 54-62; Col. 5, Lines 62-67)

71. **As to claim 74** (Original) (incorporating the rejection in claim 72), please refer to claim **3** as set forth above accordingly.

72. **As to claim 75** (Original) (incorporating the rejection in claim 72), please refer to claim **50** as set forth above accordingly.

73. **As to claim 76** (Original) (incorporating the rejection in claim 72), please refer to claim **68** as set forth above accordingly.

74. **As to claim 77** (Original) (incorporating the rejection in claim 72), please refer to claim **69** as set forth above accordingly.

75. **As to claim 78** (Original) (incorporating the rejection in claim 72), please refer to claim **53** as set forth above accordingly.

76. **As to claim 79** (Original) (incorporating the rejection in claim 72), please refer to claim **20** as set forth above accordingly.

77. **As to claim 80** (Original) (incorporating the rejection in claim 79), please refer to claim **22** as set forth above accordingly.

78. **As to claim 81** (Original) (incorporating the rejection in claim 78), please refer to claim **26** as set forth above accordingly.

79. **As to claim 82** (Original) (incorporating the rejection in claim 81), please refer to claim **28** as set forth above accordingly.

80. **As to claim 84** (Original) (incorporating the rejection in claim 83), please refer to claim **33** as set forth above accordingly.

81. **As to claim 85** (Original) (incorporating the rejection in claim 83), please refer to claim **34** as set forth above accordingly.

82. **As to claim 86** (Original) (incorporating the rejection in claim 72), please refer to claim **39** as set forth above accordingly.

83. **As to claim 87** (Original) (incorporating the rejection in claim 86), please refer to claim **43** as set forth above accordingly.

84. **As to claim 88** (Original) (incorporating the rejection in claim 87), please refer to claim **44** as set forth above accordingly.

85. Claims 2, 48, 65, and 73 are rejected under 35 U.S.C. 103(a) as being unpatentable over Murphy in view of Paxon and Liao and further in view of Szafron et al., (*LexAGen: An Interactive Incremental Scanner Generator, May 1990, Software-Practice and Experience*) (hereinafter 'Szafron')

86. **As to claim 2** (Original) (incorporating the rejection in claim 1), Murphy, Paxon, and Liao do not disclose the scanner is capable of being used for at least one of a compiler front end, a syntax-highlighting editor, a text stream editor, a parser and a parser-filter.

However, in an analogous art of *LexAGen: An Interactive Incremental Scanner* Generator, Szafron discloses the scanner is capable of being used for at least one of a compiler front end, a syntax-highlighting editor, a text stream editor, a parser and a parser-filter (e.g., Sec. of The User Interface)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Paxon into the Murphy's system to further provide the scanner is capable of being used for at least one of a compiler front end, a syntax-highlighting editor, a text stream editor, a parser and a parser-filter in Murphy-Paxon-Liao system.

The motivation is that it would further enhance the Murphy-Paxon-Liao's system by taking, advancing and/or incorporating Szafron's system which offers significant advantages for constructing scanners incrementally and that specifications can be executed anytime for validation testing; *LexAGen* Specifications are expressed and

entered interactively in a restricted BNF format as once suggested by Szafron (e.g., Summary, 1st Par. – 2nd Par.)

87. **As to claim 65** (Original) (incorporating the rejection in claim 64), please refer to claim **2** as set forth above accordingly.

88. **As to claim 73** (Original) (incorporating the rejection in claim 72), please refer to claim **2** as set forth above accordingly.

Conclusion

89. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192
June 27, 2008

/Tuan Q. Dam/

Supervisory Patent Examiner, Art Unit 2192